下载APP

16 | 循环与文件目录管理：如何实现文件的批量重命名？

2021-03-18 尹会生

Python自动化办公实战课 进入课程 >

讲述：尹会生
时长 19:55 大小 18.25M

你好，我是尹会生。

作为产品经理 / 运营，你经常需要做竞品调研，来跟自家产品对比优劣。这时，你就需要下载大量与该类产品相关的图片，并进行整理归类。而归类操作中，必须要走的一步就是对这些图片进行重命名。

还有你在搜集资料时，往往会从网络中下载大量不同格式的文件，比如电子书、视频、音频等，为了方便归纳整理，你也需要对文件进行重命名。

通过我例举的这两个场景，你应该发现了，这些需求都是把**批量改名和网络功能**结合，而且还需要 Mac 和 Windows 系统支持重命名。那怎么来实现批量重命名的操作呢?

如果你还记得上节课的内容，肯定会说，使用 PowerShell 就可以了。不过你要是对 PowerShell 相关知识掌握得扎实的话，也会记得我说过，PowerShell 适合在 Windows 平台下独立运行的临时性任务。所以在非 Windows 系统，以及需要和网络下载功能结合的需求上，它就没有 Python 的兼容性好。

那么今天这节课，我会带你回到 Python，使用 Python 来实现文件的批量重命名。

如何使用 os 库实现文件批量重命名

还是老规矩，我们学习一个新的功能，首先要学习它所需要的 Python 库和函数。

例如，我把友商的产品介绍图片，通过网络批量下载后，存放在“/Users/user1/Desktop/pic”文件夹中，这些文件名称长短不一，那我希望能从数字 1 开始，为它们批量重命名。

在第 15 节课我就说了，**重命名是一种贴近操作系统层面的操作**。因此在 Python 标准库中，我把“文件”和“操作系统”作为关键字，从官方文档中找到了“os”库，os 库中包含了操作系统相关的操作。通过“os”库，你就可以轻松取得三个批量重命名必备操作，那就是目录中的文件名、文件名后缀处理以及文件改名。

那我先把批量重命名的代码提供给你，让你更直观地理解运行过程，然后再来帮你分析这三个操作是怎么通过 os 库实现的。

复制代码

```
import os
# 保存图片的目录
file_path = "/Users/user1/Desktop/pic"
# 需要批量重命名的扩展名
old_ext = ".jpg"
# 取得指定文件夹下的文件列表
old_names = os.listdir(file_path)
# 新文件名称从1开始
new_name = 1

# 取得所有的文件名
for old_name in old_names:

    # 根据扩展名，判断文件是否需要改名
    if old_name.endswith(old_ext):

```

```
        # 完整的文件路径
        old_path = os.path.join(file_path, old_name)

        # 新的文件名
        new_path = os.path.join(file_path, str(new_name)+".JPG")

        # 重命名
        os.rename(old_path, new_path)

        # 文件名数字加1
        new_name = int(new_name)+1

# 显示改名后的结果
print(os.listdir(file_path))

```

这段代码会把"/Users/user1/Desktop/pic"目录中“jpg”扩展名的文件进行重命名，把它们分别命名为“1.jpg”，“2.jpg”，“3.jpg”，以此类推。

那批量改名的这三个操作具体是怎么实现的呢？其实是通过 os 库中的三个函数来实现的，分别是 listdir()、path.join() 、rename()。这是你在这节课要掌握的重点，我也会着重讲解。

第一个是 **listdir() 函数**，它的功能是**打印指定目录下的文件名称。如果再**给这个函数指定一个参数 file_path，那么它会把 file_path 中的所有文件名称以一个列表的类型返回。使用列表类型方便后续迭代，便于进行单个文件改名。

为了只对“.jpg”扩展名的文件改名，我使用了 endswith() 函数对列表进一步筛选，过滤掉不需要改名的文件。

第二个是 **path.join() 函数**，它可以连接路径和文件名，从而得到一个带有完整路径的文件名称格式。这里我要给你着重强调一下，改名操作必须要指定文件正确的路径，因此改名前必须要进行路径和文件名的连接。

最后是改名函数 **rename() 函数**，它的两个参数分别为改名前文件的路径 + 文件名和改名后文件的路径 + 文件名，均为 path.join() 函数处理过的完整文件名称。通过 rename() 函数改名后，文件名称会自动变为新的文件名称。

将上面三个操作放入 for 循环语句中，就能实现批量重命名的功能。

这一段代码虽然功能正常，但是继续增加新功能时，必然要再增加新的代码，如果只是按照执行的前后顺序把**多个不同功能的代码**写入一个文件，它的可读性会变差，别人阅读你写的代码就会有障碍。

就像我们从超市购买的各类商品一样，你一定会把调料放在厨房、把鸡蛋放在冰箱、把袜子放在衣柜，对商品分门别类放置，绝不会按照购买的时间顺序摆放在你的房间中。

同理，代码的摆放位置，也不应该按照执行顺序依次存放。正确的做法是，你要把每一组功能相同或相近的代码写入到一个函数中，并把该功能中经常变动的部分作为函数的参数，乃至整个脚本的参数，这样才能给有多个功能的脚本带来更好的阅读体验。代码的整洁程度高，也为你排除代码的故障带来更高的效率。

那么接下来，我就教你怎么重构批量改名脚本，提高代码的可读性。

重构程序

首先，我来带你学习一下什么是重构代码，以及怎么重构代码。

重构代码是指在代码可以正常实现的前提下，为了提高它的可维护性，需要对代码的结构进一步调整。就像你需要定期收拾房间一样，代码也需要进行维护。特别是经常修改和添加新的功能的代码，它的逻辑结构会像你炒菜之后的厨房一样，越来越混乱，为了代码和代码之间的逻辑关系更清晰，你需要掌握如何调整代码的结构。

我来给你举个例子，比如我在批量改名的程序中又增加了新的需求，要求将改名的路径和扩展名从变量赋值改为从命令行参数赋值。这样就不用进入到脚本中修改代码了。

根据这个新的需求，你会发现，当前的代码有 3 个地方需要调整。

第一个是**代码的结构层次**需要调整。当前的代码只包含了一个批量改名的功能，当你再为代码增加命令行参数处理功能时，新的代码和当前代码放在一起，会破坏改名功能的完整性，这时候，你就可以把每个功能单独放在一个函数中，通过函数来让一个功能和另一个功能代码相互独立。

第二个是**代码开始执行的位置需要调整。**由于函数定义的代码块会在函数调用以后才运行，但是根据 Python 的语法，你必须将函数定义写在函数调用的上方，这就导致了代码

开始执行的位置出现在文件的中间和结尾。所以我需要一个更明显的标记，告诉阅读代码的人，从该位置开始阅读代码，该位置才是代码执行的第一条语句，而不是让阅读的人从代码文件开头一行以后的找程序的入口。

第三个是**命令行参数的处理需要调整。**由于 Python 默认是不去处理命令行的参数的，因此我们需要增加一个专门处理命令行参数的函数，来读取用户输入的正确参数，而对错误的参数，则需要提示用户。

这三个地方的调整，我会依次采用函数、内置变量和命令行参数来实现对代码的重构，我来依次带你看一下优化的具体代码。

封装到函数

为了让代码结构逻辑更加工整，我把每一个独立的功能都放入到单独的函数中。每个函数组成的语句块，就像自然段一样，将一整篇文章，按照功能进行了划分。由于当前的代码只有批量改名这一个功能，所以我就把改名功能的所有代码都放到一个函数当中。

封装函数的时候，一个是要考虑功能的完整性，另一个要考虑函数用到的其他变量如何与它进行交互。调用函数时使用参数，就是函数和其他变量交互最好的办法。

对于批量改名这一功能，主要交互的对象有两个，它们是批量改名的路径，以及要修改的文件扩展名。所以我就把这两个对象作为改名函数 rename() 函数的参数，rename() 函数得到这两个参数后，会按照函数的定义，把这两个参数传入 rename() 函数中实现改名的逻辑，对文件进行批量改名。封装之后的核心代码如下：

复制代码

```
def rename(file_path, old_ext):
  # 批量改名的代码
  ... ...

rename("/Users/user1/Desktop/pic", ".jpg")

```

这段代码实现的功能和没有重构之前完全相同，都是对指定目录的指定扩展名文件进行批量重命名。但是在代码结构上，要比直接在文件实现的代码逻辑更清晰，可以看到，改名功能被放在函数定义中，执行的时候就可以直接调用 rename() 函数。

将改名功能封装为函数的好处就是，代码更工整了，新的功能也可以继续采用函数的形式添加到当前代码中。比起把所有代码按执行顺序都写在一个文件中，这样的格式会让你更容易区分开代码中的每一个功能。

明确执行位置

把批量改名的功能封装为函数之后，对程序的执行顺序也会带来一些变化。我把前后变化给你做个对比：

封装函数之前，程序的执行顺序是导入库之后依次执行。

封装为函数之后，执行顺序就变为导入库之后，就开始执行 rename() 函数的调用。

当这个脚本再陆续添加新的函数的话，那么找到哪一行是脚本第一个执行的命令，就非常麻烦了。因此在 Python 中有一个参考 C 语言设置代码入口的方法，让你能快速定位代码是从哪一行开始执行的。这个方法就是通过对内置变量“**name**”的值进行判断，判断它是不是和字符串“**main**” 相等。

在 Python 中，执行代码的方式有两种。

1. 一种是单独运行，也就是用 Python 加脚本的名称方式运行。
2. 另一种方式是把.py 结尾的脚本文件作为自定义的模块使用“import”关键字导入，导入后通过“模块. 函数 ()”的格式运行。

如果一个脚本文件独立运行，那么它的内置变量“**name**”的值就是“**main**”，通过“if **name** == "**main**"” 的判断，结果必然为 True，则该判断逻辑下的代码块就会执行。如果作为模块导入，那么“**name**”的值就是 False，则不被执行。

我们可以把函数的调用全部放入 “if **name** == "**main**"”语句块中，这样就可以指定这条 if 语句作为代码单独运行的入口，既方便你快速找到入口对程序进行修改，又方便你把它作为其他程序的模块进行导入。

我把实现对“**name**”变量判断的脚本写在下方，你可以对照代码学习。

复制代码

```
1
```

```
def rename():
    ... ...
def func1():
    ... ...
def func2():
    ... ...
def func3():
    ... ...

# func1() # 在__name__之外执行，不推荐

if __name__ == "__main__":
    func3()
    rename("/Users/edz/Desktop/pic", ".jpg")
    func1()
    func2()
```

在代码中，我定义了 4 个函数，对于四个函数的调用，都放在了 “if **name** == "**main**"” 语句块中。在使用这种方式设置程序入口时，有两点需要你特别注意。

一方面，这种设置方法是人为指定程序入口，因此你需要把代码中所有函数调用都放在 if 语句块下，这样才能实现作为入口的功能。虽然放在 if 语句块之外也可以运行，但函数调用写在 if 语句块之外，就很容易给代码阅读带来障碍。

另一方面，使用“**name**” 作为入口的判断变量，只能在单独运行的时候才为"**main**"，如果使用 Python 交互方式执行，就无法对“**name**” 变量进行判断。

我们通过指定代码的入口，让程序的逻辑更加清晰。那么接下来就是为这段代码添加命令行参数，在不修改代码的前提下，通过命令行参数来设置批量改名的目录和扩展名。

命令行参数处理

使用命令行参数的优点，就是在调用脚本的时候一并传入要操作的对象，这会比修改配置文件和变量更直接。那么在原有代码基础上，我们还需要增加两个参数，也就是要操作的目录和扩展名，并使用 argparse 库实现对这两个参数的处理。

参数处理是一个比较笼统的概念，它包括参数的接收、参数数量的判断和参数的解析三个部分。 “argparse” 库是命令行解析模块，它负责在脚本运行时，接收和处理脚本执行时的参数。

首先是**参数的接收**，在本讲之前，我们执行 Python 脚本的方式是：

```
python3 脚本名称.py
```

在脚本中使用“argparse”库后，脚本能够支持在该命令后面增加参数，并在脚本内获取参数的内容。哪些参数能够被脚本处理，需要使用 argparse 库的 add_argument() 函数指定。

接下来是**参数的判断**，add_argument() 函数可以接收两种参数格式，分别是“-”和“--”，后面再跟着英文。按照惯例，一个“-”一般后面会使用单个英文字母， 两个“--”后面是完整名称。

以对目录改名的参数为例，我需要接收“-p”或“--path”两种形式的参数指定的方法是：

```
add_argument("-p", "--path", required=True, help="path to rename")
```

同时，我还为“--path”参数所在的 add_argument() 增加了两个额外的参数，一个是要求用户执行程序，必须输入“-p”或“--path”，如果执行不指定会报错的 required 参数。

另一个“-p”或“--path”参数含义的帮助信息“help”参数。

增加参数处理后，如果你没有输入完整参数，argparse 库会自报错，并提示你如何正确使用该脚本的参数。你也可以直接使用“-h”得到执行帮助。

我把参数输入不完整和通过 -h 获取帮助的执行结果，贴在下面供你学习。

```
SHELL$ python3 rename_v2.py -p /path/to/rename/files -e
usage: rename_v2.py [-h] -p PATH -e EXT
rename_v2.py: error: argument -e/--ext: expected one argument

```

```
SHELL$ python3 rename_v2.py -h
usage: rename_v2.py [-h] -p PATH -e EXT

optional arguments:
  -h, --help            show this help message and exit
  -p PATH, --path PATH  path to rename
  -e EXT, --ext EXT     files name extension, eg: jpg
```

最后是**参数的解析**，它是在参数数量正确的前提下自动完成的。完成解析后，会以“--path”参数后的英文字面“path”作为属性名称，以“--path”后面的参数，作为属性值。

比如我在取得用户参数后，就可以使用“args.path”来得到命令行“-p”参数后面参数的值，以及使用“args.ext”得到“-e”参数后面参数的值。此外，我还把这两个属性作为批量改名函数 rename() 函数的参数，这样就可以把命令行参数作为重命名函数的参数使用了。

获取命令行参数的核心代码我也为你整理了出来，放在下方供你参考：

复制代码

```
import os
import argparse

def rename(file_path, old_ext):
  """批量改名函数"""
  ... ...
def args_opt():
  """获取命令行参数函数"""

    #定义参数对象
    parser = argparse.ArgumentParser()

    # 增加参数选项、是否必须、帮助信息
    parser.add_argument("-p", "--path", required=True, help="path to rename")
    parser.add_argument("-e", "--ext", required=True, help="files name extensi

    # 返回取得的所有参数
    return  parser.parse_args()

if __name__ == "__main__":

    # args 对象包含所有参数，属性是命令行参数的完整名称
    args = args_opt()

```

```
    # 调用重命名函数，将命令行参数作为重命名函数的参数
    rename(args.path, "."+args.ext)

    # 输出改名之后的结果
    print(os.listdir(args.path))
```

通过重构后，代码的整体结构就变成了导入库、函数定义、函数调用三个部分，对经常需要变动的替换路径和扩展名，也从修改变量改为命令行参数，无论从阅读代码还是后续继续扩展代码，整体结构都要比顺序执行代码逻辑更清晰。

小结

最后让我来为你总结一下，这一讲我为你讲解了使用 Python 如何实现批量改名，以及如何对越写越长的代码进行重构。

批量改名属于操作系统中的文件相关操作，这类功能在编程语言中往往会提供事先定义好的编程接口，无需自己实现从应用层到操作系统的全部代码，建议你在遇到这类需求时，先从标准库中搜索相关模块，再从第三方库搜索，尽量避免手工编写，提高工作效率。

为了让批量改名的脚本逻辑更清晰，也更方便执行，我对代码还进行了三个方面的重构：

1. 通过使用函数增加代码的逻辑性。
2. 通过“**name**”变量增加了程序入口，便于你直接找到程序开始执行的位置。
3. 通过增加命令行参数，让你不用修改代码，就能实现函数的参数的修改。

增加程序的可读性、提高执行便利性也能为以后编写代码效率提升带来改进，这些改进会在后续章节代码越来越多的时候起到更明显的效果。

我把这节课的相关代码都放在了 GitHub 上，供你学习参考。

思考题

最后我来为你留一道比较有趣的思考题，你能否通过命令行为代码指定两个参数，当这两个参数为整数时，脚本自动计算这两个参数的“和”和“差”，并将执行结果打印到屏幕上。

欢迎把你的思考和想法写在评论区，我们一起交流讨论。此外，你还可以点击课程详情页的“戳我进群”，然后扫描二维码，加入我们的课程读者群，我也会在群里为你解疑答惑。我们下节课再见！

提建议

© 版权归极客邦科技所有，未经许可不得传播售卖。 页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇 15 | PowerShell脚本：如何实现文件批量处理的自动化？

下一篇 17 | 不同操作系统下，如何通过网络同步文件？

精选留言 (2)

写留言

武明

2021-03-30

听了老师的讲解才明白那个name变量到底是什么回事儿，虽然之前会用，但是却不知道怎么回事儿，谢谢老师

展开

作者回复: 加油

Soul of the Drago...

2021-03-19

老师，我在调用命令行参数时，出现了“An exception has occurred, use %tb to see the full traceback.
SystemExit: 2” 这样的显示，请问是什么原因?

展开

作者回复: 这种情况应该只会在python交互模式里出现，原因是argparse对交互模式或jupyer支持不够完善导致的，改成用脚本方式运行就可以了。